

FlowME: Lattice-based Traffic Measurement

Petko Valtchev *, Omar Mounaouar *, Omar Cherkaoui *, Alexandar Dimitrov * and Laurent Marchand †

*Department d'informatique, UQAM, Montreal,

Email: petko.valtchev@uqam.ca

† Ericsson Montreal, Canada

Abstract—Flow-based traffic measurement is a very challenging problem: Managing counters for each individual traffic flow in hardware resources knowingly struggle to scale with high-speed links. In this paper we propose a novel lattice theory-based approach that improves flow-based measurement performances and scales by keeping the number of the maintained hardware counters to a minimum (result mathematically established in the paper). The crucial contribution of the lattice is to map the computational semantics of the packet processing to user requests for traffic measurement thus allowing for a better-informed and focused counter assignment. An implementation over an Openflow switch, FlowME, was developed and evaluated upon its memory usage, performance overhead, and processing effort to generate the minimal solution. Experimental results indicate a significant decrease in resource consumption.

I. INTRODUCTION

Network traffic measurement is an essential activity that allows network managers to get the visibility required for daily operations and network evolution planning. Tools to observe per-flow traffic must scale with a wide spectrum of applications, flows and queries while maintaining the performance of the underlying hardware, achieving accurate traffic measurements and operating at wire speed [1]. Conventional solutions like NetFlow sample traffic and send per-flow statistics to a remote server to exploit in user applications, thus incurring inaccurate statistics and intensive resource and network bandwidth usage. Recent works [2], [3] in application-aware traffic measurement use also prior knowledge about users requirements, i.e. user queries, to achieve adaptive measurements but they require dedicated packet classification mechanisms to carry out the measurement task.

A significant drawback of current solutions is they largely ignore the computational structure of the packet processing and the induced query-to-flow associations. We regard these associations as crucial and believe that only a structure that

correctly expresses them has the richness and flexibility to support the search for an optimal counter assignment. Thus, we turn to concept lattices and formal concept analysis (FCA) [4].

Our lattice-based traffic measurement method, FlowME, enables fine-grain querying of the network traffic flows and extracting of the query-bound flow measurements. Figure 1 illustrates its main components: As a main supporting structure, a hierarchy of high-level, flows-to-matchfields abstractions, the *concepts*, is constructed (a) and each query is mapped to a unique node thereof, its *target* concept (comprising the answer set of flows). The hierarchy, or the *concept lattice*, factors out commonalities in the answer sets by turning them into concepts. Targets induce a sub-hierarchy where each node - called *projection* - corresponds to an intersection of answer sets. In order to avoid the redundancies in the resulting family of sets, each flow is mapped to a minimal projection, its *ground* (b). We show that by assigning a counter to every projection, a system of counters is obtained which is both minimal in size and allows all the queries to get a precise answer (c). As a result, in FlowME the hardware counters are kept for disjoint sets of flow entries (instead of passively monitoring all flows). Moreover, the memory usage is further reduced by focusing only on flows matching user queries.

The contributions of this work are as follows.

- We propose four algorithms for constructing/maintaining the concept lattice and its projection substructure (section II). The projection algorithms are original methods that underlie the central task of partitioning the global set of flows into disjoint subsets (to be assigned a counter each).
- We prove that the number of counters established in this way is minimal w.r.t. the requirements of: (i) answering all active queries, and (ii) assigning a single counter to a flow (Theorems 3.10 and 3.11 in Section III).
- The FlowME solution can be used for flow-based measurement in a wide range of network devices and is expected, in particular, to enable effective monitoring in Openflow switches. Its implementation over an Openflow Pizzabox switch largely outperforms a per-flow counter assignment at a reasonable computational cost.

In the remainder of the paper, we first present our lattice construction and updating algorithms (section II). In section III the mathematical foundations of our solution are summarized and its efficiency/minimality are proven. Section IV presents the major components of our implementation as well as its performance evaluation results. We discuss related work in

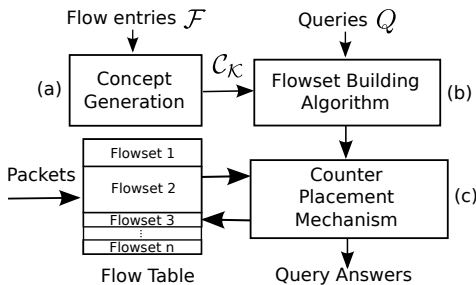


Fig. 1. Architecture Overview

Section V and conclude in Section VI.

II. ALGORITHMS

Following a novel statement of the traffic measurement problem, we introduce a set of easy-to-implement algorithms for building/maintaining lattices and counter structures and illustrate them with an example. The global workflow is illustrated in Figure 2. The lattice building algorithm (a) outputs a structure that hierarchically organizes flow entries. Flowset partition identification (b) and extraction (c) algorithms find optimal groups of flow entries based on user queries.

TABLE I
NOTATIONS

\mathcal{F}	Set of supported flow entries	\mathcal{H}	Set of matchfield values
\mathcal{M}	Flow-to-matchfield incidence	\mathcal{K}	A Context $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$
$\mathcal{C}_{\mathcal{K}}$	Context \mathcal{K} concept set	I_c	Concept c Intent
E_c	Concept c Extent	\hat{c}	Concept c parent concepts
$v(c)$	Concept c query vector	Q	Set of user queries
$g(c)$	Flows grounded in c	$ \cdot $	The cardinality of a set
$t(c)$	Queries targeted at c	T	Set of Target concepts
P	Set of Projection concepts	G	Set of Ground concepts

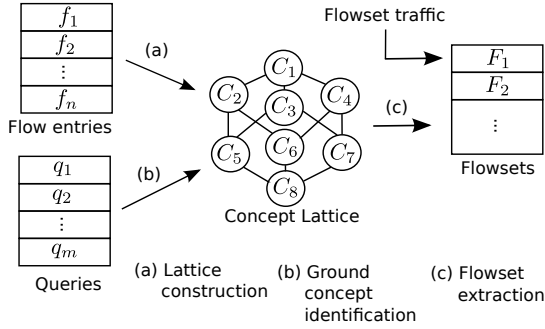


Fig. 2. Lattice based traffic measurement overview

A. Definitions

Below, a summary of the notions underlying our approach is provided (see [4], [5] for a complete coverage).

- A flow entry f is defined by its set of matchfields $\{h_1, h_2, \dots, h_n\}$ and is assigned a counter to be updated whenever a packet matches f . Let \mathcal{F} be the set of all flows installed in a specific switch while F is flowset, i.e., an arbitrary set of flows (as in [2]). Let \mathcal{H} be the set of all matchfield values from \mathcal{F} .
- A user query $q \in Q$ is a sequence of regular expressions on flow matchfields. Here, we assume a query is merely a set of matchfield values.
- A context $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$ associates \mathcal{F} to \mathcal{H} via an incidence relation $\mathcal{M} \subseteq \mathcal{F} \times \mathcal{H}$. In \mathcal{K} two *image* operators $'$ lift \mathcal{M} to the set level: flow/matchfield sets are mapped into the set of incident matchfields/flows (quantification is universal).
- A concept is a pair (F, H) , where $F \in \wp(\mathcal{F})$ (*extent*) and $H \in \wp(\mathcal{H})$ (*intent*) are s.t. $F = H'$ and $H = F'$. The

set $\mathcal{C}_{\mathcal{K}}$ of all concepts in \mathcal{K} is partially ordered by extent inclusion:

$$(F_1, H_1) \leq_{\mathcal{K}} (F_2, H_2) \Leftrightarrow F_1 \subseteq F_2, (H_2 \subseteq H_1).$$

$\langle \mathcal{C}_{\mathcal{K}}, \leq_{\mathcal{K}} \rangle$ is a complete lattice, as meets \wedge and joins \vee are defined for arbitrary concept sets. The precedence $<_{\mathcal{K}}$, transitive reduction of $\leq_{\mathcal{K}}$, induces the Hasse diagram of the lattice.

- Compositions $''$ of complementary images $'$ are closure operators on $\wp(\mathcal{F})$ and $\wp(\mathcal{H})$, respectively. The families of extents, $\mathcal{C}_{\mathcal{K}}^f$, and of intents, $\mathcal{C}_{\mathcal{K}}^h$, are closed by \cap . Thus, for a set A of flows (of matchfields) A'' is the smallest extent (intent) comprising A .
- T is the set of target concepts: for a query q its target is $\gamma(q) = (q', q'')$; P is the set of projections, i.e., the meets of non-empty set of targets: $c_p = \bigwedge T_p$, $T_p \subseteq T$; G is the set of ground projections: for a flow f its ground is $\mu(f) = \min(\{(F, H) \in P | f \in F\})$.

B. Problem statement

The traffic measurement optimization problem consists in, given a set of flows \mathcal{F} to monitor and a set of users queries Q , finding the minimal partition of \mathcal{F} while being able to answer all user queries. In this settings, the number of partitions in the classical approaches equals the number of flows, therefore, traffic measurement resources are maximal in all usage contexts. Our running example, $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$, is shown in Table II.

TABLE II
INPUT CONTEXT

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
f_0	x			x			x		x	
f_1	x			x	x		x			x
f_2		x				x		x		
f_3		x				x		x		x
f_4	x						x		x	
f_5	x				x		x			x
f_6			x			x			x	x
f_7			x			x		x		x

h_1	- Ingress Port = 1	h_6	- IPv4 src = 132.208.130/32
h_2	- Ingress Port = 2	h_7	- IPv4 src = 10/8
h_3	- Ingress Port = 3	h_8	- IPv4 dst = 10/8
h_4	- MAC src = MAC ₁	h_9	- IPv4 dst = 132.208.130.1
h_5	- MAC dst = MAC ₁₂	h_{10}	- Layer 4 dst port = 21

C. Lattice construction

Algorithm 1 is a version of *NextNeighbor* in [5], (p.35). It constructs the lattice from the top concept $(\mathcal{F}, \mathcal{F}')$ down to the bottom one $(\mathcal{H}', \mathcal{H})$, by generating the children of the current concept (F, H) . To that end, it first produces the extents of a larger set of sub-concepts by intersection of F with the images of all matchfields outside of H . It then connects as children of (F, H) only the maximums of the resulting set (and enqueues these for further processing). For instance, at concept $c_8 = (\{f_0, f_1, f_4, f_5\}, \{h_1, h_7\})$ in Figure 3, the following four extents are generated: \emptyset (by intersections with $h'_2, h'_3, h'_6,$

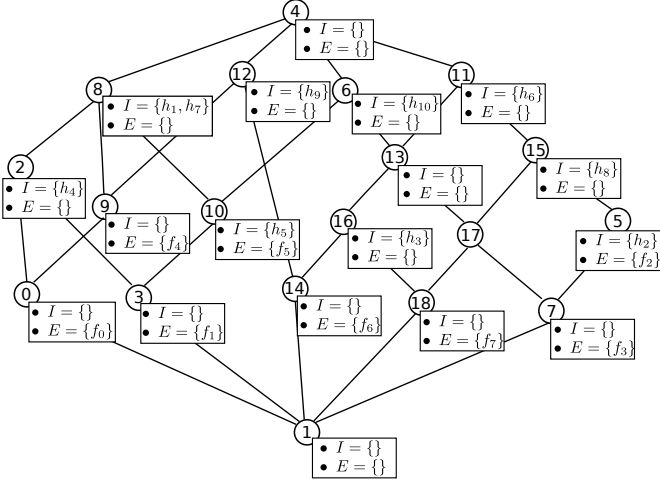


Fig. 3. Concept lattice of the input context: reduced concept intents/extents are provided to increase readability (objects are inherited upwards and attributes downwards)

Algorithm 1: Lattice construction algorithm

```

input :  $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$ 
output: Set of linked concepts  $C$ 
1  $ConceptQueue \leftarrow \{(\mathcal{F}, \mathcal{F}'')\}$ ;
2 while  $ConceptQueue \neq \emptyset$  do
3    $c = (F_c, H_c) \leftarrow ConceptQueue.pop()$ ;
4    $Children \leftarrow \emptyset$ ;
5   foreach  $h$  in  $\mathcal{H} - H_c$  do
6      $F_h = F_c \cup h$ ;
7     if  $\exists \bar{c} \in Children$  s.t.  $E_{\bar{c}} = F_h$  then
8        $E_{\bar{c}} \leftarrow E_{\bar{c}} \cup \{h\}$ ;
9     else
10       $Children \leftarrow Children \cup \{(F_h, H_c \cup \{h\})\}$ 
11   $\tilde{c} \leftarrow \max(Children)$ ;
12   $C \leftarrow C \cup \tilde{c}$ ;

```

and h'_8), $\{f_0, f_1\}$ (with h'_4), $\{f_1, f_5\}$ (with h'_5 and h'_{10}), and $\{f_0, f_4\}$ (with h'_9). The latter three are maximal, hence they are the extents of children concepts for c_8 (c_2 , c_{10} , and c_9 , respectively).

D. Flowset partition identification

The ultimate goal is to split \mathcal{F} into disjoint sets to be assigned a single counter each. Assume a query set $Q = \{q_i\}_{i=1..5}$ with $q_1 = \{h_{10}\}$, $q_2 = \{h_2, h_6, h_8\}$, $q_3 = \{h_1\}$, $q_4 = \{h_1, h_4, h_7\}$ and $q_5 = \{h_7\}$. Given the concept set C_K and Q , Algorithm 2 parses C_K to identify T , P and G . Projection computation is supported by a bitvector whose value for $c = (F, H)$ reflects the queries satisfied by flows in F . Formally, the query vector $v(c)$ is an N-bit string indicating which q_i are matched by H :

$$v((F, H))[i] = \begin{cases} 1, & \text{if } q_i \subseteq H \\ 0, & \text{otherwise} \end{cases} \quad 1 \leq i \leq N$$

Algorithm 2: Flowset partition identification algorithm

```

input : A list of concepts  $C$ ,
        A set of queries  $Q = (q_1, \dots, q_i, \dots, q_n)$ 
output: Target, projection and ground sets  $(T, P, G)$ 
1  $Sort(C)$ ;
2 foreach  $c$  in  $C$  do
3   for  $q_i$  in  $Q$  do
4     if  $q_i \subseteq I_c$  then
5        $v(c)[i] \leftarrow 1$ ;
6        $T \leftarrow T \cup \{c\}$ ;  $Q \leftarrow Q - \{q_i\}$ ;
7    $v(c) \leftarrow v(c) \cup \bigcup_{\bar{c} \in c^-} v(\bar{c})$ ;
8   if  $|v(c)| > \max_{\bar{c} \in c^-} (|v(\bar{c})|)$  then
9      $P \leftarrow P \cup \{c\}$ ;
10  if  $|I_c| = I$  then
11    for  $p \in P$  do
12      if  $v(p) = v(c)$  then
13         $G \leftarrow G \cup \{p\}$ ;  $break()$ ;

```

TABLE III
QUERY VECTORS VALUES

Query vector	Concepts	Query vector	Concepts
00000	$c_4, c_{11}, c_{12}, c_{15}$	00111	c_0, c_2
10000	$c_6, c_{13}, c_{14}, c_{16}, c_{17}, c_{18}$	10101	c_{10}
01000	c_5	11000	c_7
10111	c_3	00101	c_8, c_9
11111	c_1		

First, the concepts list C is sorted in decreasing order of extent sizes (line 1), to ensure the first concept whose intent matches a $q \in Q$ is its target (line 4). Matched q are removed from the list (line 6). In our example, the algorithm outputs the targets c_6, c_5, c_8, c_2 and c_8 , for q_i ($i = 1..5$), respectively. Then, the value of $v(c)$ is finalized (line 7): the local part (targeted queries, line 5) is merged with the inherited parent values (see results in Table III). Projections are concepts whose query vectors have more 1s than any of their respective parent ones (line 8). For instance, c_{10} has three 1s, more than its parents c_8 (one) and c_6 (two), hence it is a projection (as meet of the targets c_6 and c_8). Finally, the ground concept of a $f \in \mathcal{F}$ is the projection with the same query vector as the flow concept (f'', f) (lines 10-13). Table IV provides the flow-to-ground mapping of our example.

E. Flowset extraction

The optimal partition is composed by the target concept extents: $\Phi_g = \{F | \exists (F, H) \in G\}$. A hardware counter is assigned to each flowset in Φ_g , i.e., a total of $m = |\Phi_g|$ counters. And since ground intents are disjoint, $m \leq |\mathcal{F}|$ with = reached with exclusively singleton flowsets. With G from Table IV, f_6 and f_7 share a common counter, whereas the remaining flows get a dedicated counter each.

TABLE IV
FLOW-TO-GROUND CONCEPT MAPPING

Flow	Ground	Flow	Ground	Flow	Ground	Flow	Ground
f_0	c_2	f_1	c_3	f_2	c_5	f_3	c_7
f_4	c_8	f_5	c_{10}	f_6	c_6	f_7	c_6

F. New flow entry insertion

Algorithm 3: Lattice update : Add a flow

```

input : Added flow entry  $f_n$ 
input/output: Concept lists  $(C, T, P, G)$ 
1  $C_n \leftarrow \emptyset; M \leftarrow \emptyset;$ 
2 foreach  $c = (E_c, I_c)$  in  $C$  do
3    $H \leftarrow I_c \cap f'_n;$ 
4   if  $H = I_c$  then
5      $c \leftarrow (E_c \cup \{f_n\}, I_c); M \leftarrow M \cup \{c\};$ 
6   else if  $\nexists \bar{c} \in C_n \cup M$  s.t.  $I_{\bar{c}} = H$  then
7      $C_n \leftarrow C_n \cup \{c_n = (E_c \cup \{f_n\}, H)\};$ 
8      $\hat{c} \leftarrow \hat{c} \cup \{c_n\}; c_n^\sim \leftarrow c_n^\sim \cup \{c\};$ 
9      $UpdateOrder(c_n, c, C_n, M);$ 
10     $UpdateStatus(c_n, c, T, P, G);$ 
11  $c_p \leftarrow Lookup(P, v(\mu(f_n))); //Ground\ of\ f_n$ 
12  $g(c_p) \leftarrow g(c_p) \cup \{f_n\};$ 
13  $G \leftarrow G \cup \{c_p\};$ 
14  $C \leftarrow C \cup C_n;$ 

```

Assume a new flow f_8 with $f'_8 = \{h_2, h_7, h_9\}$ is added to the initial context. Algorithm 3 implements the schema in [6] to update the lattice \mathcal{L}_K to the lattice of $\mathcal{K}_n = (\mathcal{F}_n, \mathcal{H}, \mathcal{M}_n)$, where $\mathcal{F}_n = \mathcal{F} \cup \{f_n\}$ and $\mathcal{M}_n = \mathcal{M} \cup \{f_n\} \times f'_n$. The basic task consists of producing all intersections of the new flow image f'_n with intents from \mathcal{C}_K . For intersections that are intents in \mathcal{K} , the extent of the underlying concept (qualified as *modified*) is updated with f_n (line 5). For instance, c_{12} yields $I_{c_{12}} \cap f'_8 = I_{c_{12}}$, thus its extent is updated with f_8 (see the updated lattice in Figure 4). The only other modified is c_4 . Figure 4 presents the updated lattice in Figure 3. Observe that concept numbers are IDs: concepts with the *same numbers* as in Figure 3 have the *same intents*.

An intersection missing in C_n^h triggers the creation of a new concept (only the first time). The intent of c_n is the intersection itself, while the extent is the extent of the generating concept (alias the *genitor*) plus f_n (line 7). In our example, c_{19} , c_{20} , c_{21} , and c_{22} are the new concepts with genitors c_8 , c_9 , c_5 , and c_1 , respectively. Among them, c_{22} is the flow concept of f_8 .

To update the $<_{\mathcal{K}}$ links, first c_n and its genitor are linked as a parent and a child, respectively (line 8). Then, the children of c_n are chosen among the already identified part of the new and modified concept sets (line 9).

The UpdateStatus method, detailed by Algorithm 4, establishes the status of the new concept c_n (target, projection, ground, none) and updates that of its genitor c . In our example, c_{19} matches $q_5 = \{h_7\}$, thus, q_5 will be re-targeted from the

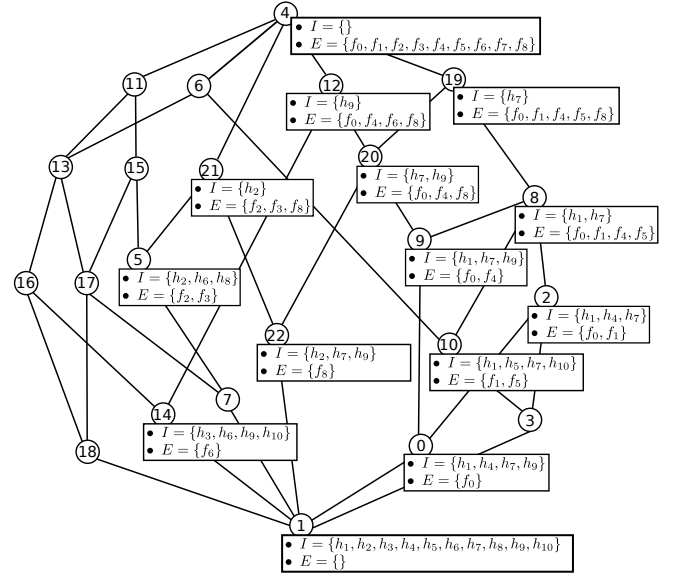


Fig. 4. The concept lattice of the extended context: only full intents/ extents of relevant concepts are drawn.

Algorithm 4: UpdateStatus: Update support structure

```

input/output: concepts  $c$ ,  $c_n$  (the new and its genitor)
    Concept sets  $(T, P, G)$ 

1 foreach  $q_i \in t(c)$  do
2   if  $q_i \subseteq H$  then
3      $t(c_n) \leftarrow t(c_n) \cup \{q_i\}; t(c) \leftarrow t(c) - \{q_i\};$ 
4      $v(c_n)[i] \leftarrow 1;$ 
5 if  $t(c_n) \neq \emptyset$  then  $T \leftarrow T \cup \{c_n\};$ 
6 if  $t(c) = \emptyset$  then  $T \leftarrow T - \{c\};$ 
7 if  $|v(c_n)| > \max_{\bar{c} \in \hat{c}_n} (|v(\bar{c})|)$  then
8    $P \leftarrow P \cup \{c_n\};$ 
9   if  $|v(c)| = |v(c_n)|$  then
10      $P \leftarrow P - \{c\};$ 
11   if  $g(c_n) \neq \emptyset$  then
12      $g(c_n) \leftarrow g(c); g(c) \leftarrow \emptyset;$ 
13      $G \leftarrow G \cup \{c_n\} - \{c\};$ 

```

genitor c_8 to the new concept c_{19} (line 3). The projection test (line 7) follows the one in Algorithm 2. Next, if the query vector of c_n has the same number of 1s as the vector of c (line 7), the latter is no more a projection (line 8). In this case, all flows grounded at c are re-grounded at c_n (lines 11-13).

Finally, in a post-processing step (lines 11-13, Algorithm 3) the ground concept of the new flow f_n is established as the projection $c_p \in P$ with the same query vector as the flow concept $\mu(f_n)$ (line 11). In our example, $v(\mu(f_8)) = v(c_{22}) = 00001$ is the same as the projection c_{19} query vector. The new flow f_8 is grounded to c_{19} (line 13).

To sum up the restructuring: The new target concept list is $T_n = T \cup \{c_{19}\}$, the new projections $P_n = P \cup \{c_{19}\}$, and the

new grounds $G_n = G \cup \{c_{19}\}$.

Our solution comprises algorithms for removing of flows as well as for adding/removing queries or individual matchfield values that we do not provide here for space limitation reasons. All our algorithms follow a similar computational schema: they perform a traversal of the concept lattice whereas the most effort-intensive processing task for a concept boils down to one set-based operation on the intents/extends of each parent/child. Hence the algorithmic complexity is $O(|\mathcal{C}_K| * |\mathcal{F}| * (|\mathcal{H}| + |\mathcal{F}|))$ [6]. Below, we show how the richness and regularity of the lattice and its counter-related substructure translate into algorithmic efficiency and optimality of the solution.

III. PROOFS

We establish below the correctness of our algorithms (Properties 3.1 to 3.9 and Theorem 3.10) and the minimality of the proposed counter assignment (Theorem 3.11).

A. Lattice construction and maintenance

Algorithm 1 is rooted in the following results [7]: (1) For a concept (F, H) with children (F_i, H_i) , the faces $H_i - H$ are pairwise disjoint, and (2) $\forall h \in H_i - H, h' \cap F = F_i$. Hence it constructs all (F_i, H_i) from (F, H) by producing all possible $h' \cap F$ for matchfields $h \in \mathcal{H} - H$ and partitioning those h into the faces of (F, H) . Non face matchfields from $\mathcal{H} - H$ generate smaller intersections that are filtered out.

Algorithm 3 (lattice update) transforms \mathcal{C}_K into \mathcal{C}_{K_n} with the emphasis on the completion of the intent family $\mathcal{C}_{K_n}^h$ to $\mathcal{C}_{K_n}^h$ (since $\mathcal{H}_n = \mathcal{H}$). Indeed, existing intents provably remain valid in \mathcal{K}_n : $\mathcal{C}_{K_n}^h \subseteq \mathcal{C}_{K_n}^h$. Moreover, the new intents are pairwise intersections of f'_n with elements from \mathcal{C}_K^h :

Property 3.1: $\mathcal{C}_{K_n}^h = \mathcal{C}_K^h \cup \{\{f_n\}' \cap H \mid H \in \mathcal{C}_K^h\}$.

Correspondingly, the extents of concepts in \mathcal{C}_{K_n} have only two possible forms: F or $F \cup \{f_n\}$ where $F \in \mathcal{C}_K^f$.

A new intent generates a new concept: Although it may be produced more than once, a canonical generator, the *genitor*, exists that holds a two-fold bound to the new concept, i.e., both through its intent and extent:

Property 3.2: For a $(F, H) \in \mathcal{C}_{K_n}$, s.t. $H \in \mathcal{C}_{K_n}^h - \mathcal{C}_K^h$, $\exists (F_g, H_g) \in \mathcal{C}_K$ s.t. $H = H_g \cap f'_n$ and $F = F_g \cup \{f_n\}$.

As a corollary, the genitor intent is the closure of the new one in \mathcal{K} : $H'' = H_g$.

Modified concept intents H are provably s.t. $H \subseteq \{f_n\}'$ (closed in \mathcal{K}). Hence in \mathcal{K}_n the respective extents H' comprise f_n . Observe that genitor and modified have intents that are the closures of their $H \subseteq \{f_n\}'$, hence they are the maximal concepts to produce it. Thus, they will be the first ones to reach along the top-down breadth-first traversal of the lattice. Finally, the adjustment of the new precedence among concepts in \mathcal{C}_{K_n} is skipped here (interested readers are directed to [7]).

B. Measurement support construction

Observe $\gamma(q)$ is the maximal $(F, H) \in \mathcal{C}_K$ s.t. $q \subseteq H$ while F is the set of all flows f satisfying q ($q \subseteq f'$). Moreover, $\mu(f)$ is well defined: it is the meet of the targets of queries satisfied by f ($\mu(f) = \bigwedge \{\gamma(q) \mid q \in Q; q \subseteq f'\}$).

The main tasks in Algorithm 2 are detecting all $\gamma(q)$ (the highest concept (F, H) with $q \subseteq H$) and propagating the targeted q downwards in the lattice. These q are stored in the bitvectors $v()$ for further projection tests. Now, a projection c is exactly the meet of the targets of queries in $v(c)$:

Property 3.3: $c \in P$ iff $c = \bigwedge \{\gamma(q_i) \mid v(c)[i] = 1\}$.

As a corollary, the projection extent is the intersection of the target ones ($F = \bigcap \{E_{\gamma(q_i)} \mid v(c)[i] = 1\}$). Then, c is maximal for $v(c)$ and thus can be recognized by comparing its bitvector to those of parent concepts:

Property 3.4: $c \in P$ iff $\forall \bar{c} \in \hat{c}, v(\bar{c}) \neq v(c)$.

As it is readily shown that as a function $v()$ is monotonously non increasing w.r.t. \leq , the property may be recast in terms of cardinalities: $|v(c)| > \max_{\bar{c} \in \hat{c}} (|v(\bar{c})|)$.

Finally, G is tested by comparing $v(c)$ to bitvectors of flow concepts:

Property 3.5: For a $c = (F, H)$, $c \in G$ iff $\exists f \in F$ s.t. for $\bar{c} = (f'', f')$, $v(\bar{c}) = v(c)$.

Moreover, as in our specific case, no flow has a subset of another flow's matchfields, $\forall f \in \mathcal{F}, f'' = f$. Thus flow concepts are exactly those with singleton extents.

C. Measurement support maintenance

To show that T , P and G are correctly transformed into T_n , P_n and C_n , respectively, by Algorithm 4, observe that for $c \in \mathcal{C}_K$ $v(c)$ keeps its value in \mathcal{C}_{K_n} .

Property 3.6: For a concepts $c = (F, H) \in \mathcal{C}_{K_n}$, if $H \in \mathcal{C}_K^h$ then $v_n(c) = v(\bar{c})$ where $\bar{c} = (H', H)$.

The reason is $v(c)$ only depends on H and Q which remain stable in \mathcal{K}_n . Thus, the function $v_n()$ evolves from $v()$ by merely computing the values for new concepts in C_n (value propagation matches the downward generation of C_n).

Now, T_n may depart from T as some $q \in Q$ may change targets ($\gamma(q) \neq \gamma_n(q)$). Clearly, $\gamma_n(q)$ can only be a new concept in \mathcal{K}_n , whereas $\gamma(q)$ is its genitor in \mathcal{K} .

Property 3.7: For a query $q \in Q$ s.t. $\gamma(q) \neq \gamma_n(q)$, $\bar{c} = \gamma_n(q)$ is a new concept in \mathcal{C}_{K_n} while $c = \gamma(q)$ is its genitor.

This follows from the minimality of H'' among intents comprising H . Thus, with $c = (F, H)$ and $c = (F_n, H_n)$, we show that $H = H''_n$ (hence the genitor status) in \mathcal{K} . Indeed, assuming $H \neq H''_n$, we deduce $H''_n \subset H$ (*) since $q \subseteq H$ (recall $q'' = H$) and H_n is the closure of q in \mathcal{K}_n (minimal intent comprising q). Yet since $q \subseteq H_n \subset H''_n$, (*) would contradict the minimality of $H = q''$ in \mathcal{K} .

P_n evolves from P along two separate scenarios: (1) as with T_n , a new concept may become projection by eclipsing its genitor in P_n ; and (2) a new concept may become the infimum for a set of query targets with no equivalent in P . Recall that projections are identified within P by $v()$ (Property 3.3). In other terms, in case one, the infimum c of a set of targets ($v(c)$) evolves to a different concept \dot{c} (diverging intents) with the same bitvector value ($v(c) = v_n(\dot{c})$), whereas in case two, a previously nonexistent set of targets $v_n(\dot{c})$ arises.

Property 3.8: Given a $c \in P_n$, if c is not the equivalent of the projection concept $\bar{c} = \bigwedge \{\gamma(q_i) \mid v_n(c)[i] = 1\}$ in \mathcal{K} , then c is a new concept with \bar{c} as its genitor.

Assume that for some $c = (F, H) \in P_n$, the projection $\bar{c} = \bigwedge \{\gamma(q_i) | v_n(c)[i] = 1\}$ from P is such that $\bar{c} = (F_o, H_o)$ and $H_o \neq H$ (\bar{c} not an equivalent concept in \mathcal{K} , despite $v(c) = v(\bar{c})$). The latter means $F \neq F_o$, and since these are the intersection of the target extents from $v(c)$, it follows that *all those extents* have changed. As we saw previously, the only possible evolution of a target extent for a query q in \mathcal{K}_n is to increase by f_n . Consequently, their intersection F (corollary of Property 3.3) comprises f_n as well and, as H is not an intent in \mathcal{K} , c is a new concept. By the same argument, F_o can only be $F_o = F - \{f_n\}$, hence \bar{c} is the genitor of c (Property 3.2).

In case two, the new projection c is a new concept too:

Property 3.9: Given a $c = (F, H) \in P_n$, if for all projections $\bar{c} \in P$, $v_n(c) \neq v(\bar{c})$, then c is a new concept.

Assuming the opposite, let $H \in \mathcal{C}_{\mathcal{K}}^h$, hence c is not a new concept ($f_n \notin F$) and thus $v_n(c) = v(c)$. Consequently there is a concept with the same bitvector value in \mathcal{K} , c itself, which further means there must be a maximal concept \bar{c} , s.t. $v(c) = v(\bar{c})$, i.e., an infimum. This contradicts the starting hypothesis.

G_n being a subset of P_n , similar evolution patterns hold: In the above case one, all flows grounded at the genitor –which vanishes from P_n , hence from G_n – must be re-grounded at the new projection c_n . In case two, no flow from \mathcal{F} could be grounded in c_n , since their flow concepts in $\mathcal{C}_{\mathcal{K}_n}$ have intents from $\mathcal{C}_{\mathcal{K}}^h$. Thus, the respective bitvectors do not change in \mathcal{K}_n , hence all such flows are grounded in c whose $v_n(c)$ existed in P . Therefore, f_n is the only candidate for case two grounds.

To sum up, in T_n , new concepts of target genitors grab targeted queries comprised in their respective intents, while genitors with no remaining queries vanish. In P_n , new concepts are tested for projection and, if positive, genitors too. In G_n , flows grounded at a shifting projection move from the genitor to the new concept. Finally, $\mu(f_n)$ is found.

D. Correctness and minimality of counter assignment

We prove that ground concept-based counter assignment is: (1) correct, and (2) of minimal cardinality. Recall that each ground $c_g \in G$ is assigned a counter whose support is the set of grounded flows denoted $g(c_g) = \{f | \mu(f) = c_g\}$. This is a unique counter assignment (uniqueness of $\mu(f)$) and w.l.o.g. we assume that each flow is grounded. Furthermore, for each $q \in Q$ the set of relevant counters compose to a sum and let the underlying total set of flows be $S(q)$. As a counter enters a query sum iff its ground is below the corresponding target, we have $\forall f \in \mathcal{F}, q \in Q, f \in S(q) \text{ iff } \mu(f) \leq \gamma(q)$.

Correctness means a $S(q)$ is the set of flows satisfying q :

Theorem 3.10: $\forall q \in Q, f \in \mathcal{F}, q \subseteq f' \text{ iff } \mu(f) \leq \gamma(q)$.

'If': follows from intent inclusion along \leq : $q \subseteq I_{\gamma(q)} \subseteq I_{\mu(f)} \subseteq f'$. 'Only if': Observe that satisfaction means $f \in E_{\gamma(q)}$ and assume, by *reductio ad absurdum*, $\mu(f) \not\leq \gamma(q)$. Then the infimum $c_{q,f} = \mu(f) \wedge \gamma(q) \in P$ (as \wedge is associative) whereas $f \in E_{c_{q,f}}$. Yet this contradicts $\mu(f) \not\leq \gamma(q)$ as then $c_{q,f} < \mu(f)$ (minimal in P to hold f).

Conversely, redundancy in $S(q)$ is excluded since a relevant flow f appears exactly once in it (through $\mu(f)$).

Minimality means no unique counter assignment among a smaller set of counters could answer all q in Q . We focus on the underlying partition of \mathcal{F} :

Theorem 3.11: Let $cpt : \mathcal{F} \rightarrow \wp(\mathcal{F})$ with $cpt(f) = F$ iff $f \in F$ and assume $|\text{ran}(cpt)| < |G|$. Then $\exists q \in Q$ s.t. $S(q)$ is not decomposable into the union of some sets from $\text{ran}(cpt)$.

By *reductio ad absurdum*, assume all $S(q)$ represent unions of $cpt(f)$ for flows f from a well-chosen set. A straightforward combinatorial argument yields $\exists f_1, f_2 \in \mathcal{F}$, s.t. $\mu(f_1) \neq \mu(f_2)$ (**) yet $cpt(f_1) = cpt(f_2)$. Yet (**) means $v(\mu(f_1)) \neq v(\mu(f_2))$ and w.l.o.g. we can assume $\exists q_a \in Q$, s.t. $q \subseteq f'_1$ but $q_a \not\subseteq f'_2$. However, this contradicts the initial hypothesis since there is no way to correctly decompose $S(q_a)$ into a union of $cpt(f)$: if $cpt(f_1)$ participates, then there is no way to remove the contribution of f_2 (subtraction not available) while otherwise, there is no way to recover the contribution of f_1 ($cpt(f_1)$ is its unique counter).

IV. IMPLEMENTATION AND RESULTS

A FlowME implementation over an OpenFlow switch was studied along three measurement axes: (1) memory cost expressed in term of number of managed counters, (2) processing effort for concept lattice generation/update, and (3) performance overhead on packet processing. We choose OpenFlow because flow entries and user queries can be pushed and retrieved from the Openflow tables of the switch. Experimental results show a huge reduction in the number of hardware counters with a reasonable overall computational effort and negligible interference on traffic.

A. Testbed design

The FlowME testbed comprises an OpenFlow Switch with per-flow counter support, a flow entry generator, a Collector and user applications that generate queries. As shown in Figure 5, FLOWME Collector gets the set of flow entries \mathcal{F} (a) installed in the Flow table of the OpenFlow switch and user queries Q (b). It calls upon lattice algorithms of the Coron FCA suite [8] to calculate/maintain the optimal flow entry partition and exploits it to place flow counter references (c). Next, traffic matching \mathcal{F} increments hardware counters (d). FlowME collector reads counter values (e), calculates query answers and sends them to user applications (f). We experimented FlowME with a variety of flow entry and query distributions.

1) *Flow entry benchmarking*: Flow entry benchmark generates up to 12 fields per entries. It is based on Flexible Rule Generator [9], a user controlled benchmarking tool for evaluating packet forwarding algorithms that generate sets of OpenFlow flow entries based on predefined matchfield distributions. We extract matchfield distributions from packet traces provided by packetlife.net. The traces are particularly interesting since packet headers contain different types of fields as MAC, VLAN, IP and transport fields. As a result, a total of 12 standard OpenFlow matchfields are used in the benchmark. We analyse packet trace headers to determine a distribution function for each matchfield. Table VI shows a

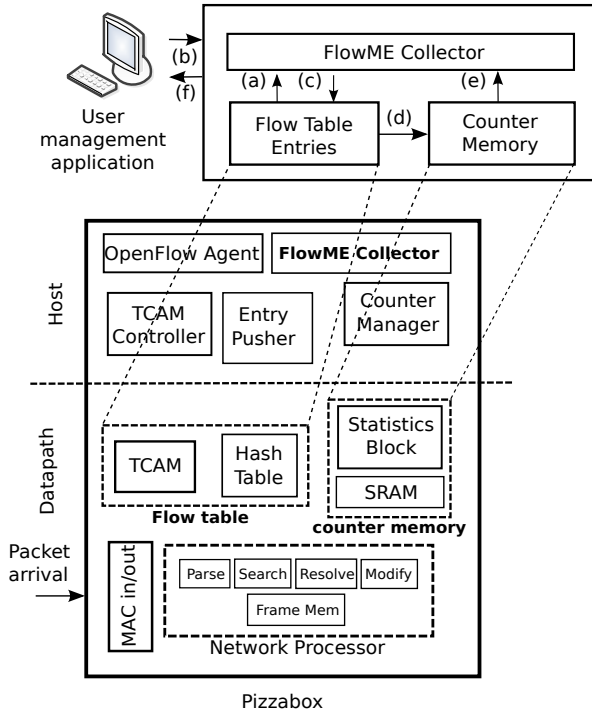


Fig. 5. FlowME testbed

set of matchfields and their value distributions. Notice that IP source and destination analysis involves the prefix distribution and the prefix length distribution.

TABLE V
PACKET TRACE MATCHFIELD VALUE DISTRIBUTION OF DENSITY $\geq 3\%$

Matchfield	Distribution
MAC src	00:40:05(39%), 08:00:07(13%), 00:60:08(19%)
MAC dst	00:60:08(33%), FF:FF:FF(37%), 00:40:05(19%)
Ethertype	0x8100(98)%
VLAN id	32(56%), 104(17%), 108(4%), 6(6%)
IP protocol	0x06(80%), 0x11(6%), 0x01(13%)
TOS	0(96%), 192(3%)
L4 src port	2212(41%), 1815(26%), 2388(11%), 8(4%)
L4 dst port	1815(53%), 2212(18%), 2388(8%), 3314(4%)

2) *Query benchmarking*: The second benchmark generates application queries. A query covers a set of flow entries whose size depends on how many matchfields get a non wildcard value. In our experimental study, we generate user queries with the same matchfield value distribution as flow entries, in which we inserted some wildcarded values (a specific percentage for each matchfield). Moreover, we force each query to cover at least one flow entry. To that end, we first extract n flow entries from \mathcal{F} and then insert a specific percentage of wildcards in each matchfield, thus yielding a set of n queries.

B. Switch implementation

We use the 100 Gig OpenFlow implementation over EZchip's network processor introduced in [10], and we add flow counter support. In our solution, each flow in memory is split across a set of pipelined tables (see Figure 6). Each

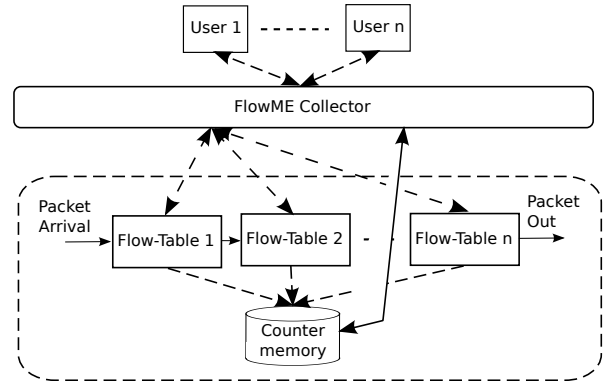


Fig. 6. Implementation over an OpenFlow switch

table is implemented using TCAM for flow matchfield classification and Hash table for flow entry instruction and counter reference storage. Flow tables report counters in a continuous memory space where SRAM supports first 8192 counters and RLDRAM supports the rest. In a typical scenario, as the one shown in Figure 5 and 6, when a packet is received, relevant header fields are parsed by a Parse engine, and a key is built. A lookup is performed by a Search engine in the TCAM against flow entry matchfields, and if they match, the TCAM provides an index of the matching flow entry. In order to retrieve counters and instructions associated to that entry, a second lookup is performed in Hash table based on the entry index. A Resolve engine receives the entry and processes its hardware counter reference and instructions (prepared for an eventual execution). The processing repeats on subsequent flow tables. At the very end of flow identification, Resolve engine sends a hardware counter increment command to the Statistics Block via a dedicated routine.

C. Memory cost

Per-flow traffic measurement tools manage an individual counter for each traffic flow processed by the system (the set \mathcal{F}) and report individual flow statistics to a centralized collector. In those systems, the counter number N_c evolves linearly with $|\mathcal{F}|$ since, in practice, for each flow $f \in \mathcal{F}$, the system may need different traffic metrics, e.g. the number of matching packets or their total size. In contrast, our solution relies on aggregated counters, so we ran FlowME with the above benchmarks and observed the N_c value. The evolution of the number of counters to maintain in order to answer a set of user queries Q of size N_Q is depicted in Figures 7, 8 and 9. In Figure 7, query field values are composed of 10% exact match values and 90% wild-cards. In Figures 8 and 9, queries are more specific with 50% and 90% exact match values, respectively. For example, the first experiment shows that for $N_Q = 1000$, N_c is significantly lower than the 10000 per-flow counters of the base-line solution (949 to 3555, depending on wild-card distribution). As a general trend, we observe that as less specific queries cover more flow entries each, the number of (minimal) intersections is higher and thus the counter set grows larger. Let M be the available

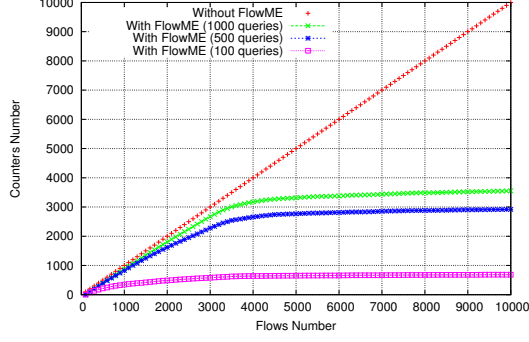


Fig. 7. Number of managed counters for varying query set sizes. Query field value distribution: 10% exact-match 90% wildcard

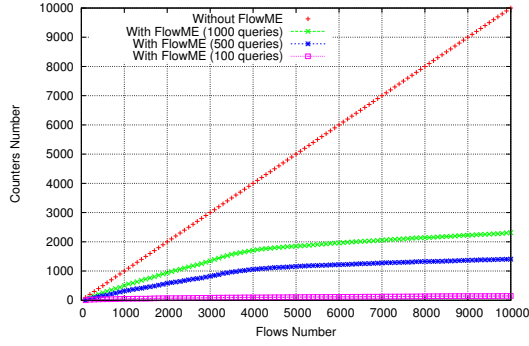


Fig. 8. Number of managed counters for varying query set sizes. Query field value distribution: 50% exact-match 50% wildcard

memory size reserved for traffic measurement, M_u the set of managed counter registers in a specific measurement period. M_u is bounded by the number of flow entries installed in the switch. In traditional flow-based measurement schemes, $M_u = |\mathcal{F}|$. In contrast, FlowME is an application-aware measurement tool, hence $|\mathcal{F}|$ is the worst case that occurs only if the ground intersections generated by Q split \mathcal{F} into its singleton components. Table VI illustrates relative memory consumption for our solution in experiment one (Figure 7 with $N_Q = 1000$).

TABLE VI
COMPARISON IN MEMORY USAGE ($N_Q = 1000$)

Technique	# of Counters	SRAM usage
Per-Flow	8192	100%
FlowME	3555	43%

D. Lattice structure generation time

The flexibility of FlowME is assessed by measuring the update time for a new flow entry. Two main operations are monitored: (1) lattice update and (2) ground concept identification and flow partition extraction. In our experiment, 10000 flow entries are split into groups of 100 to be sent for lattice update at subsequent steps of the incremental process. Figure 10 shows the number of flow entries added at each step

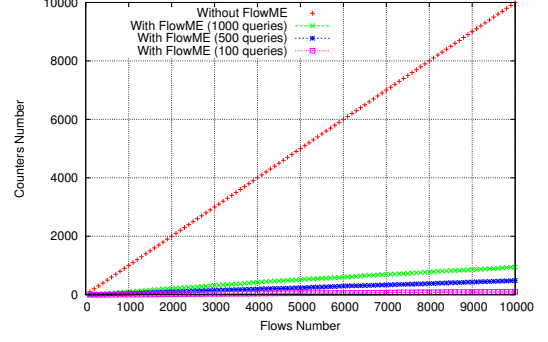


Fig. 9. Number of managed counters for varying query set sizes. Query field value distribution: 90% exact-match 10% wildcard

in the experimentation period. For instance, in the first 8s, 1000 flows are added. Overall, it took a total of 5min30s to build the lattice and identify flow entry partitions incrementally.

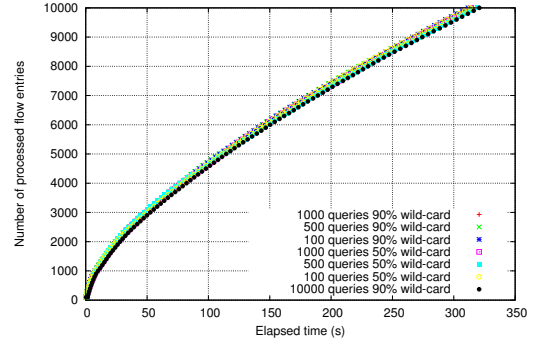


Fig. 10. Structure incrementation time: A group of 100 flow entries is added to the structure at each iteration

E. Packet processing performance

The main challenge in traffic measurement is to minimize the impact on packet processing performance (time in clock cycles). In our testbed using a 400MHz network processor, the average total packet processing time is 2447ns. Since FlowME focuses only on flows covered by user queries, the remaining flows are automatically accelerated. As indicated above, we implemented a routine for the packet resolving stage that increments flow counters based on the reference retrieved at the searching stage. Processing time is now measured by placing start/end timestamps around the routine. According to the experimental outcome, the increment routine is performed in 9 clock cycles. Hence, for an unmanaged flow packet, FlowME lowers the total processing time by 22.5ns.

V. RELATED WORK

The closest approach from the literature is the ProgME [2] traffic measurement tool. ProgME reflects application requirements through a rich query language where \cap , \cup and \setminus are used to compose queries from simpler ones. To answer a set of queries, ProgME decomposes them into a set of

disjoints flowsets, and assigns a counter to each one. In that, it does not rely on predefined flows. In contrast, our flows, and queries for that matters, are defined as simple conjunctions of matchfield values. However, the same set-theoretic operators on the answer sets of flows corresponding to queries are successfully simulated by our ground projection-based partition. In particular, the set of flows grounded in a projection represent the set-theoretic difference between the projection extent and the union of all extents of smaller projections. As a result, our own partition of \mathcal{F} comprises a local partition of each answer set of flow entries. Now, our main advantage over ProgME lays in the (proven) minimality of counter-assignment solution: The lattice structure ensures none of the implicit set operations is redundant whereas the disentangling algorithm in ProgME lacks such a result.

AutoFocus [11] is a tool for offline hierarchical traffic analysis whose goal is complementary to ours. Like ProgME, it doesn't use predefined flows but rather discovers them. To that end, it mines hierarchies of frequent generalized values for each matchfield and combines them into a global multi-dimensional structure. The structure comprises both the most significant and some deviant flows. As the authors themselves admit, the approach boils down to mining frequent generalized patterns on multiple dimensions. In comparison, our lattice contains the frequent *closed* patterns of matchfield values from \mathcal{F} which is a strict subset of all frequent patterns [7].

Current flow-based monitoring and collection systems like Cisco Netflow, FlowScan [12] and sFlow [13] track all flow statistics continuously at a specific sampling rate. This generates a large number of transactions and a management bandwidth usage proportional to the number of flows, regardless of real application needs. The comparison of FlowME to a flow-based measurement technique (section IV-C) shows the huge reduction in the number of managed flows.

Finally, since the introduction of metered traffic groups by the ISO accounting model [14] a number of architectures based on that notion were proposed in IETF internet RFCs (e.g. [15]). Identification of traffic groups remains an open problem and is typically solved by network operations personnel. FlowME is a significant step toward its automation.

For an in-depth coverage of the traffic measurement field readers are referred to [2].

VI. CONCLUSION

We presented FlowME, a lattice-based traffic measurement solution that, we believe, is a significant contribution to the field. Its mathematically founded approach amounts to partitioning the set of flow entries into a minimal number of subsets, each assigned a hardware counter. The main advantages thereof, efficiency in statistic computation and optimal resource usage, have been experimentally confirmed through an implementation over an OpenFlow switch: The results show both a significant reduction in the number of hardware counters (up to a factor of 10) and excellent performances. Moreover, our algorithmic methods are easy to implement while highly flexible and adaptable to a wide range of contexts.

Within a broader scope, a crucial benefit of our solution is its predictability: A preliminary assessment of the resources required by a user request is enabled in order to ensure their consumption respects the acceptable limits, i.e., 10%. Overall, our approach will enable user control over the set of statistics in OpenFlow 1.3 [16] instead of fixing them within the standard. Furthermore, the genericity of the mathematical solution makes it particularly suitable to future network protocols with open sets of packet fields (CDN [17], NDN [18], etc.).

Finally, the high versatility of the lattice-based framework enables large variations in the problem settings. For instance, in the shorter term, we shall investigate the reduced substructures of the lattice as support for the counter assignment.

REFERENCES

- [1] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 323–336, Aug. 2002.
- [2] L. Yuan, C.-N. Chuah, and P. Mohapatra, "Progme: towards programmable network measurement," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 115–128, Feb. 2011.
- [3] F. Ghannadian, L. Fang, and M. J. Quinn, "Adaptive, flow-based network traffic measurement and monitoring system," U.S. Patent US 7 639 613, Dec 29, 2009.
- [4] B. Ganter and R. Wille, *Formal Concept Analysis, Mathematical Foundations*. Springer-Verlag, 1999.
- [5] C. Carpineto and G. Romano, *Concept Data Analysis : Theory and Applications*. Wiley, 2004.
- [6] P. Valtchev, R. Rouane-Hacene, and R. Missaoui, "A generic scheme for the design of efficient on-line algorithms for lattices," in *Proc. of the 11th Intl. Conf. on Conceptual Structures (ICCS'03), Dresden, Germany, July 21-25*, ser. LNCS v.2746, 2003, pp. 282–295.
- [7] P. Valtchev, R. Missaoui, and R. Godin, "Formal Concept Analysis for Knowledge Discovery and Data Mining: The New Challenges," in *Proc. of the 2nd Intl. Conf. on Formal Concept Analysis (ICFCA'04), Sydney, Australia*, ser. LNCS v.2961. Springer, 2004, pp. 352–371.
- [8] "The Coron System." [Online]. Available: <http://coron.loria.fr>
- [9] T. Ganegedara, W. Jiang, and V. Prasanna, "Frug: A benchmark for packet forwarding in future networks," in *Proc. of the 29th IEEE Intl. Perform. Comp. & Comm. Conf. (IPCCC'10)*, dec. 2010, pp. 231–238.
- [10] O. E. Ferkouss, R. B. Ali, Y. Lemieux, and O. Cherkaoui, "Performance model for mapping processing tasks to OpenFlow switch resources," in *Proc. of the IEEE Intl. Conf. on Comm. (ICC'12)*, 2012.
- [11] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proc. of the 2003 Conf. on Appl., technol., architect., and protocols for comp. comm. (SIGCOMM'03)*. New York, NY, USA: ACM, 2003, pp. 137–148.
- [12] D. Plonka, "Flowscan: A network traffic flow reporting and visualization tool," in *Proc. of the 14th USENIX Conf. on System administration*, ser. LISA'00. Berkeley, CA, USA: USENIX Assoc., 2000, pp. 305–318.
- [13] P. Phaál, S. Panchen, and N. McKee, "Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," RFC 3176, Internet Engineering Task Force, sep 2001. [Online]. Available: <http://www.ietf.org/rfc/rfc3176.txt>
- [14] *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework*, ISO Norm ISO/IEC 7498-4:1989, 1989.
- [15] N. Brownlee, C. Mills, and G. Ruth, "Traffic flow measurement: Architecture," RFC 2722, Internet Engineering Task Force, oct 1999. [Online]. Available: <http://www.ietf.org/rfc/rfc2722.txt>
- [16] "Openflow switch specification version 1.3," Open Networking Foundation, Jun 2012.
- [17] J. Dille, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *Internet Computing, IEEE*, vol. 6, no. 5, pp. 50–58, sep/oct 2002.
- [18] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. of the 5th Intl. Conf. on Emerging networking experiments and technologies*, ser. CoNEXT'09. New York, NY, USA: ACM, 2009, pp. 1–12.